A Routing Protocol for Packet Radio Networks

Shree Murthy and J.J. Garcia-Luna-Aceves
Computer Engineering
University of California
Santa Cruz, CA 95064
shree, jj@cse.ucsc.edu

Abstract

We present a new distance-vector routing protocol for a packet radio network. The new distributed routing protocol, WRP, works on the notion of second-to-last hop node to a destination. WRP reduces the number of cases in which a temporary routing loop can occur and also provides a mechanism for the reliable transmission of update messages. The performance of WRP has been compared quantitatively by simulations with that of distributed Bellman-Ford (DBF), DUAL (a loop-free distance-vector algorithm) and an ideal link-state algorithm (ILS) which represents the state of the art of Internet routing, in a highly dynamic environment. The simulation results indicate that WRP is the most efficient of the algorithms simulated in a wireless environment.

1 Introduction

With the recent proliferation of laptop and portable computers, and the development of wireless network interfaces, host mobility is becoming an important issue. An efficient routing protocol is necessary to communicate directly with the participating computers in a highly dynamic environment in which the hosts are mobile.

The routing protocols used in multihop packet-radio networks implemented in the past [1, 2, 11] were based on shortest-path routing algorithms that have been typically based on the distributed Bellman-Ford algorithm (DBF) [3]. According to DBF, a node knows the length of the shortest path from each neighbor node to every network destination and this information is used to compute the shortest path and next node in the path to each destination. An update message contains a vector of one or more entries, each of which specifies as a minimum, the distance to a given destination. A major performance problem with DBF is that it takes a very long time to update the routing tables of network nodes after network partitions, node failures, or increases in network congestion. This performance problem of DBF stems from the fact that it has no inherent mechanism to determine when a network node should stop incrementing its distance to a given destination.

Because of DBF's performance problems, most router manufacturers have opted for routing protocols based on topology broadcast such as OSPF [12]. However, there are significant differences between the wired Internet over which standard routing protocols are used today, and wireless networks. Today's networks have relatively high bandwidth and topologies that change infrequently; in contrast, wireless networks have mobile nodes and have limited bandwidth for network control. Mobility management algorithms which involve user location and hand-off management, essentially require reliable and efficient routing algorithms. For scalability with base stations and mobile nodes, these algorithms need to be distributed. Also, the routing algorithms should be flexible enough to serve as a template to carry mobility management information. In addition, since the bandwidth is limited in a wireless environment, the routing algorithms should not suffer from looping problems.

The flooding techniques used in link-state or topology broadcast protocols create excessive traffic in a multihop radio network with dynamic topology. On the other hand, the routing protocols based on DBF or modifications of DBF take a long time to converge and the frequent topology changes in a wireless network with mobile nodes make the looping problem of DBF unacceptable. Therefore, there is a need for a new routing protocol which is devoid of all these drawbacks.

In the recent past, a number of efforts have been made to address the limitation of DBF and topology broadcast in mobile wireless networks. One such effort is the DSDV protocol [5]. In this protocol, each mobile host which is a specialized router, periodically advertises its view of the interconnection topology with other mo-

[0]This work was supported in part by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175 and by the Office of Naval Research under Contract No. N-00014-92-J-1807.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **1995**	2. REPORT TYPE	3. DATES COVERED **00-00-1995 to 00-00-1995**
4. TITLE AND SUBTITLE **A Routing Protocol for Packet Radio Networks**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**10**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

bile hosts within the network thereby maintaining an up to date information about the status of the network. But, DSDV suffers from the inherent problem of synchronization. In DSDV, a node has to wait until it receives the next update message originated by the destination to update its distance table entries. Also, it uses both periodic and triggered updates for updating routing information. This could cause excessive overhead.

A distributed routing algorithm for mobile wireless networks based on diffusing computations has been proposed by Corson and Ephremides [6]. This protocol relies on the exchange of short control packets forming a *query-reply* process. It also has the ability to maintain multiple paths to a given destination. This is a destination-oriented protocol in which separate versions of the algorithm run independently for each destination. Routing is source-initiated in which routes are maintained by those sources which actually desire routes. Even though this algorithm provides multiple paths to the destination, because of the query-based synchronization approach to achieve loop-free paths, the communication complexity could be high.

Recently, a number of distributed shortest-path algorithms have been proposed [4, 7, 9, 10, 14] that utilize information regarding the length and second-to-last hop (predecessor) of the shortest path to each destination to eliminate the counting-to-infinity problem of DBF. We call this type of algorithms as *path-finding algorithms*. According to these algorithms, each node maintains the shortest-path spanning tree reported by its neighbors. A node uses this information along with the cost of adjacent links to generate its own shortest-path spanning tree. An update message exchanged among neighbors consists of a vector of entries that report updates to the sender's spanning tree; each update entry contains a destination identifier, the distance to the destination, and the second-to-last hop of the shortest path to the destination.

Path-finding algorithms are an attractive approach for wireless networks. Although they eliminate counting-to-infinity problem, they can still incur substantial temporary loops in the paths specified by the predecessor before they converge. This can lead to slow convergence, or incur substantial processing by requiring a node to update its entire routing table for each input event. To address these problems, we have proposed a path-finding algorithm, PFA, that substantially reduces the temporary looping situations [13].

The rest of this paper describes a routing protocol for a packet radio network based on PFA, which we call *wireless routing protocol* (WRP). Section 2 gives a detailed description of WRP, illustrating the key aspects of the protocol's operation. Section 3 compares the performance of WRP with that of DBF, DUAL and ILS. Finally, Section 4 presents our conclusions.

2 Wireless Routing Protocol

2.1 Overview

To describe WRP, we model a network as an undirected graph represented as $G(V, E)$, where V is the set of nodes and E is the set of links (or edges) connecting the nodes. Each node represents a router and is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. A functional bidirectional link connecting the nodes is assigned a positive weight in each direction. All messages received (transmitted) by a node are put in the input (output) queue and are processed in FIFO order. The communication links in the network are such that all update messages transmitted over an operational link are received in the order in which they were transmitted within a finite time.

A link is assumed to exist between two nodes only if there is radio connectivity between the two nodes and they can exchange update messages reliably with a certain probability of success. When a link fails, the corresponding distance entries in a node's distance and routing tables are marked as infinity. A node failure is modeled as all links incident on that node failing at the same time.

WRP is designed to run on top of the link-level protocol of a wireless network. Update messages may be lost or corrupted due to changes in radio connectivity or jamming. Reliable transmission of update messages is implemented by means of retransmissions. After receiving an update message free of errors, a node is required to send a positive acknowledgment (ACK) indicating that it has a good radio connectivity and has processed the update message. Because of the broadcast nature of the radio channel, a node can send a single update message to inform all its neighbors about changes in its routing table; however, each such neighbor sends an ACK to the originator node.

In addition to ACKs, the connectivity can also be ascertained with the receipt of any message from a neighbor (which need not be an update message). To ensure that connectivity with a neighbor still exists when there are no recent transmissions of routing table updates or ACKs, periodic update messages without any routing table changes (null update messages) are sent to the neighbors. The time interval between two such null update messages is the *HelloInterval*.

If a node fails to receive any type of message from a neighbor for a specified amount of time (e.g., three or four times the HelloInterval known as the *RouterDeadInterval*), the node must assume that connectivity with that neighbor has been lost.

2.2 Information Maintained at Each Node

For the purpose of routing, each node maintains a *distance table*, a *routing table*, a *link-cost table*, a *message retransmission list* and an *ack-status table*.

The distance table of node i is a matrix containing, for each destination j and each neighbor of i (say k), the distance to j (D^i_{jk}) and the predecessor (p^i_{jk}) reported by k.

The routing table of a node i is a vector with an entry for each known destination j which specifies:

- The destination's identifier
- The distance to the destination (D^i_j)
- The predecessor of the shortest path chosen toward j (p^i_j)
- The successor (s^i_j) of the shortest path chosen for j
- A marker (tag^i_j) used to update routing table; it specifies whether the entry corresponds to a simple path (tag^i_j = correct), a loop (tag^i_j = error) or a destination that has not been marked (tag^i_j = null).

```
Procedure Init1
when router $i$ initializes itself
do begin
    set a link state table with costs of adjacent links;
    $N \leftarrow i; N_i \leftarrow x \mid l^i_x < \infty;$
    for each ($x \in N_i$)
    do begin
        $N_i \leftarrow N \cup x; tag^i_x \leftarrow null;$
        $s^i_x \leftarrow null; p^i_x \leftarrow null; D^i_x \leftarrow \infty$
    end
    $D^i_i \leftarrow 0; s^i_i \leftarrow null; p^i_i \leftarrow null; tag^i_i \leftarrow correct$
    for each $j \in N$ call Init2($x$, $j$)
    for each ($n \in N_i$) do add (0, $i$, 0, $i$) to $LIST_i(n)$
    $x \leftarrow$ retransmission time; $y \leftarrow$ hello count;
    $z \leftarrow$ retransmission count;
    call Send
end

Procedure Init2($x$, $j$)
begin
    $D^i_{jx} \leftarrow \infty; p^i_{jx} \leftarrow null; s^i_{jx} \leftarrow null; seqno^i_{jx} \leftarrow 0$
end

Procedure Send
begin
    for each ($n \in N_i$)
    do begin
        if ($LIST_i(n)$ is not empty)
        then send messages with $LIST_i(n)$ to $n$
        empty $LIST_i(n)$
    end
end

Procedure Message
when router $i$ receives a message on link ($i$, $k$)
begin
    if ($k \notin N_i$) do
    begin
      $N_i \leftarrow N_i \cup k;$
      $l^i_k \leftarrow$ cost of new link;
      if ($k \notin N$) begin
        $N \leftarrow N \cup k; tag^i_k \leftarrow null;$
        $D^i_k \leftarrow \infty; p^i_k \leftarrow null; s^i_k \leftarrow null;$
        for each $x \in N_i$ do call Init2($x$, $k$)
      end
      for each ($i$, $k$, $l^i_k$) do
        send update(0, $k$, $D^i_k$, $p^i_k$)
    end
    reset HelloTimer;
    for each entry ($u^k_j$, $j$, $RD^k_j$, $rp^k_j$) | $i \neq j$
    do begin
        if ($j \notin N$)
        then begin
          if ($RD^k_j = \infty$) then delete entry
          else begin
            $N \leftarrow N \cup j;$
            for each entry $x \in N_i$ call Init2($x$, $j$)
            $tag^i_j \leftarrow null$; call DT
          end
        end
        else begin
          $tag^i_j \leftarrow null;$
        end
    end
    for each entry ($u^k_j$, $j$, $RD^k_j$, $rp^k_j$) left | $i \neq j$
    do case of $u^k_j$
        0:  call Update($j$, $k$)
        1:  call ACK($j$, $k$)
    end
    call Send
end

Procedure Create_RList($seqno$)
begin
    $seqno \leftarrow seqno + 1; NeighborSet \leftarrow N_i$
    $bitmap[] \leftarrow 0; RetransmissionTimer \leftarrow x$
    add updates to RList
end

Procedure Delete_RList($seqno$)
begin
    set $bitmap[seqno] \leftarrow 1; delete \leftarrow 1$
    for all $n \in N_i$ begin
        if ($bitmap[seqno] = 0$) $delete \leftarrow 0$;
    end
    if ($delete = 1$) delete RList[seqno] end

Procedure Update_RList($seqno$)
begin
    reset RetransmissionTimer
    send update $RList[seqno]$;
end

Procedure Clean_RList ($seqno$)
begin
    for all entries in $RList$
        delete $RList[seqno]$;
end

Procedure Connectivity
when HelloTimer expires
begin
    $HelloCount[k] \leftarrow HelloCount[k] + 1;$
    if ($HelloCount[k] < y$) then reset HelloTimer;
    else begin
      $N_i \leftarrow N_i - k$
      call Delete_RList($k$)
      $l^i_k \leftarrow \infty$
      $tag^i_k \leftarrow null$
      delete column for $k$ in distance table
      update routing table
    end
end

Procedure TimeOut($i$, $k$)
when RetransmissionTimer expires
begin
    RetransmissionCounter $\leftarrow$ RetransmissionCounter - 1;
    if (RetransmissionCounter $<$ z)
      call Update_RList($k$)
    else begin
      $N_i \leftarrow N_i - k$
      call Delete_RList($k$)
      $l^i_k \leftarrow \infty$
      $tag^i_k \leftarrow null$
      delete column for $k$ in distance table
      update routing table
    end
end

Procedure DT
when distance table update has to be done
begin
    $D^i_{jk} \leftarrow l^i_k + D^k_j; p^i_{jk} \leftarrow p^k_j;$
(2) for all neighbors $b$
    do begin
        if k is in the path from i to j in
           the distance table through neighbor $b$
        then $D^i_{jb} \leftarrow D^i_{kb} + D^k_j; p^i_{jb} \leftarrow p^k_j$
    end
end
```

Figure 1: Protocol Specification

The link-cost table of node i lists the cost of relaying information through each neighbor k, and the number of periodic update periods that have elapsed since node i received any error-free messages from k.

The cost of a failed link is considered to be infinity. The way in which costs are assigned to links is beyond the scope of this specification. As an example, the cost of a link could simply be the number of hops, or the addition of the latency over the link plus some constant bias. The cost of the link from i to k (i, k) is denoted by l^i_k.

The message retransmission list (MRL) specifies one or more retransmission entries, where the m^{th} entry consists of the following:

- The sequence number of an update message
- A retransmission counter that is decremented every time node i sends a new update message
- An *ack-required* flag (denoted by a^i_{km}) that specifies whether node k has sent an ACK to the update message represented by the retransmission entry
- The list of updates sent in the update message

The above information permits node i to know which updates of an update message have to be retransmitted and which neighbors should be requested to acknowledge such retransmission. Node i retransmits the list of updates in an update message when the retransmission counter of the corresponding entry in the MRL reaches

```
Procedure ACK($n$)
when router $i$ receives an ACK on link $(i, k)$
begin
    call Delete_RList($n$);
    RetransmissionCounter $\longleftarrow$ z;
end

Procedure Update($i, k$)
when router $i$ receives an update on link $(i, k)$
begin
    send ACK to neighbor $k$
    RetransmissionCounter $\longleftarrow$ z;
    RetransmissionTimer $\longleftarrow$ x;
(0) begin
        update=0;
        $RTEMP^i \longleftarrow \phi$;
        $DTEMP^{i,b} \longleftarrow \phi$ for all neighbors $b$
(1)     for each triplet $(j, D^k_j, p^k_j)$ in $V^{k,i}$, $j \neq i$ do
            call procedure DT
(3)     begin
            if there are $b$ and $j$ such that
              $(D^i_{jb} < D^i_j)$ or $((D^i_{jb} > D^i_j)$ and $(b = s^i_j))$
            then call RT_Update
        end
(4)     begin if $(RTEMP^i \neq \phi)$ then
            for each neighbor $b$ do begin
                for each triplet $t = (j, D^i_j, p^i_j)$ in $RTEMP^i$
                do begin
                    if $b$ is not in the path from $i$ to $j$
                    then $DTEMP^{i,b} \longleftarrow DTEMP^{i,b} \cup t$;
                end
                send $DTEMP^{i,b}$ to neighbor $b$;
            end
    end
end
```

```
Procedure RT_Update
when routing table has to be updated
begin
    find minimum of the distance entries $DT_{min}$
    if $(D^i_{j s^i_j} = DT_{min})$ then $ns \longleftarrow s^i_j$
    else $ns \longleftarrow b \mid \{b \in N_i$ and $D^i_{jb} = DT_{min}\}$;
    $x \longleftarrow j$;
    while $(D^i_{x\,ns} = Min\{D^i_{xb}\ \forall\ b \in N_i\}$
        and $D^i_{x\,ns} < \infty$ and $tag^i_x = null)$
    do $x \longleftarrow p^i_{x\,ns}$;
    if $(p^i_{x\,ns} = i$ or $tag^i_x =$ correct)
    then $tag^i_j \longleftarrow$ correct  else $tag^i_j \longleftarrow$ error
    if $(tag^i_j =$ correct) then begin
      if $(D^i_j \neq DT_{min}$ or $p^i_j \neq p^i_{j\,ns})$ then begin
        $seqno \longleftarrow seqno + 1$;
        add (0, $j$, $DT_{min}$, $p^i_{j\,ns}$, $seqno$) to $LIST_i(x)\ \forall x \in N_i$;
        call Clean_RList($seqno$)
        call Create_RList($seqno$)
      end
      $D^i_j \longleftarrow DT_{min}$; $p^i_j \longleftarrow p^i_{j\,ns}$; $s^i_j \longleftarrow ns$
    end
    else begin
        if($D^i_j < \infty$) then begin
          $seqno \longleftarrow seqno + 1$;
          add (0, $j$, $\infty$, null, $seqno$) to $LIST_i(x)\ \forall x \in N_i$;
          call Clean_RList($seqno$)
          call Create_RList($seqno$)
        end
        $D^i_j \longleftarrow \infty$; $p^i_j \longleftarrow$ null; $s^i_j \longleftarrow$ null
    end
end
```

Figure 2: Protocol Specification (Cont..)

zero. The retransmission counter of a new entry in the MRL is set equal to a small number (e.g., 3 or 4).

2.3 Information Exchanged among Nodes

In WRP, nodes exchange routing-table update messages (which we call "update messages" for brevity) that propagate only from a node to its neighbors. An update message contains the following information:

- The identifier of the sending node.
- A sequence number assigned by the sending node.
- An *update list* of zero or more updates or ACKs to update messages. An update entry specifies a destination, a distance to the destination, and a predecessor to the destination. An ACK entry specifies the source and sequence number of the update message being acknowledged.
- A *response list* of zero or more nodes that should send an ACK to the update message.

In the event that the message space is not large enough to contain all the updates and ACKs that a node wants to report, they are sent in multiple update messages. An example of this event can be the case in which a node identifies a new neighbor and sends its entire routing table.

The response list of the update message is used to avoid the situation in which a neighbor is asked to send multiple ACKs to the same update message, simply because some other neighbor of the node sending the update did not acknowledge.

The first transmission of an update message must ask all neighbors to send an ACK, of course, and this is accomplished by specifying the "all-neighbors address," which consists of all 1's.

When the update message reports no updates, the "empty address" is specified; this address consists of all 0's and instructs the receiving nodes not to send an ACK in return. This type of update message is used as a "hello message" from a node to allow its neighbors to know that they maintain connectivity, even if no user messages or routing-table updates are exchanged.

As we explain subsequently, an ACK entry refers to an entire update message, not an update entry in an update message, in order to conserve bandwidth.

2.4 Routing-Table Updating

Figures 1 and 2 specify important procedures of WRP used to update the routing and distance tables.

A node can decide to update its routing table after either receiving an update message from a neighbor, or detecting a change in the status of a link to a neighbor.

When a node i receives an update message from its neighbor k, it processes each update and ACK entry of the update message in order.

In WRP, a node checks the consistency of predecessor information reported by *all* its neighbors each time it processes an event involving a neighbor k. In contrast, all previous path-finding algorithms [4, 10, 14] check the consistency of the predecessor only for the neighbor associated with the input event. This unique feature of WRP accounts for its fast convergence after a single resource failure or recovery as it eliminates more temporary looping situations than previous path-finding algorithms.

2.4.1 Processing an Update

To process an update from neighbor k regarding destination j, the distance and the predecessor entries in the distance table are updated. A flag (tag) is set to specify that this entry in the table has been changed. A unique feature of WRP is that node i also determines if the path to destination j through any of its other neighbors $\{b \in N_i | b \neq k\}$ includes node k. If the path implied by the predecessor information reported by node b includes node k, then the distance entry of that path is also updated as $D^i_{jb} = D^i_{kb} + D^k_j$ and the predecessor is updated as $p^i_{jb} = p^k_j$. Thus, a node can determine whether or not an update received from k affects its other distance and routing table entries.

To update its distance and predecessor for destination j (Procedure RT_Update), node i chooses a neighbor p that has reported routing information such that:

- The path from p to j (which is implied by the predecessor information reported by p) does not include node i
- $D^i_{jp} \leq D^i_{jx}$ for any other neighbor x, and $D^i_{yp} \leq D^i_{yx}$ for any other neighbor x and for every node y in the path from i to j.

The above means that node i chooses node p as its successor to a destination j if that neighbor appears to offer a smallest-cost loop-free path to j and all the intermediate nodes in the path to j.

When node i sends an update message, it updates its ack-status table and message retransmission list. For each destination j for whom there is an update being reported, node i sets the ack-required flag for all its neighbors. It also adds an entry in the update-retransmission list containing the sequence number given to the update message, and starts the retransmission timer for the entry.

2.4.2 Sending New and Retransmitted Update Messages

Node i sends a new update message after processing updates from its neighbors or detecting a change in a link to a neighbor. Whenever node i sends a new update message, it must

- Decrement the retransmission counter of all the existing entries in the MRL
- Delete the updates in existing entries in the MRL that are included in the new update message
- Add an entry in the MRL for the new update message

When the list of updates of a MRL entry is emptied by the transmission of a new update message, node i erases that entry from the MRL.

When the retransmission counter for a retransmission entry m in the MRL expires, node i sends an update message with a new sequence number, an update list containing the list of updates of the retransmission entry, and a response list specifying those neighbors who did not acknowledge the update message earlier (i.e., every neighbor k for whom $a^i_{km} = 1$). The retransmission counter of existing entries in the MRL is not modified.

Note that, based on the above retransmission strategy, there is no limit on the number of times node i would retransmit an update message to an existing neighbor. However, as we discuss below, node i stops considering node k as its neighbor after it fails to communicate with it for some finite amount of time.

2.4.3 Processing an ACK

An ACK entry in an update message refers to another entire update message, i.e., it acknowledges all the updates included in the update message bearing the referenced sequence number. Therefore, it is up to the node whose update message is being acknowledged to ascertain which updates are implied by a received ACK.

To process an ACK from neighbor k, node i scans its MRL for the sequence number matching the sequence number specified in the ACK received. Whenever a match is found, node i resets the ack-required flag for neighbor k; if $a^i_{pm} = 0$ for entry m and every neighbor p of node i, the retransmission entry is deleted. This scheme obtains short ACKs at the expense of additional processing.

Node i may receive an ACK for an update message whose retransmission entry has been erased after sending a more recent update message for the same destinations. In that case, node i simply ignores the ACK.

2.4.4 Handling Topology and Link-Cost Changes

To ensure that nodes know that they have connectivity even when they do not transmit user messages or routing-table updates for some time, every node i must periodically send an update message reporting no changes (hello messages). Acknowledgments are not required for such update messages, and they can be very short (e.g., a byte for control information and a byte for the node identifier, since the control information can imply that there is no sequence number, update list, or response list in the message). Alternatively, a node may retransmit an update message if it is not too long. When a node k comes up, it transmits a hello message.

Given that short periodic update messages are transmitted by every node, the failure of a link to a neighbor is detected by the lack of any user or update messages being received from that neighbor over a period of time equal to a few update-message transmission periods. Similarly, new links and nodes are detected by means update messages or user messages.

When node i receives an update or user message from node k and node k is not listed in its routing table or distance table, it adds the corresponding entry to its distance or routing table for destination k. An infinite distance to all destinations through node k is assumed, with the exception of node k itself and those destinations reported in node k's updates, if the message received from k was an update message. In addition, node i notifies node k of the information in its routing table. This information can be transmitted in one or multiple update messages that only node k needs to acknowledge.

When a link fails or a link-cost changes, node i recomputes the distances and predecessors to all affected destinations, and sends to all its neighbors an update message for all destinations whose distance or predecessor change.

2.5 Example

The following example illustrates the working of WRP. Consider a four node network shown in Figure 3(a). All links and nodes are assumed to have the same propagation delays. Link-costs are as indicated in the figure. Node i is the source node, j is the destination node and nodes k and b are the neighbors of node i. The arrows next

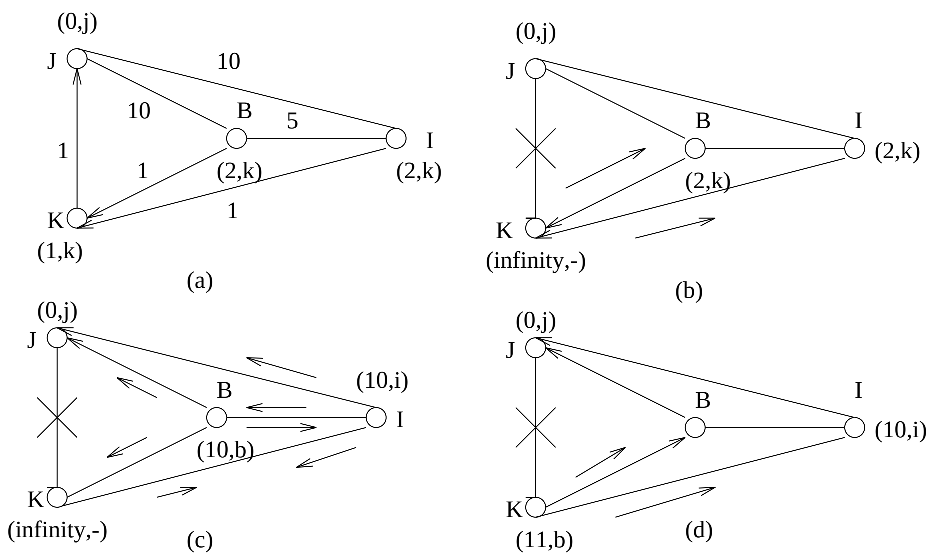


Figure 3: Example of the algorithm's operation

to links indicate the direction of updates messages and the label in parentheses gives the distance and the predecessor to destination j. Each update will be acknowledged by an ACK message from the neighbor. ACKs are not shown in the figure. The figure focuses on update messages to destination j only.

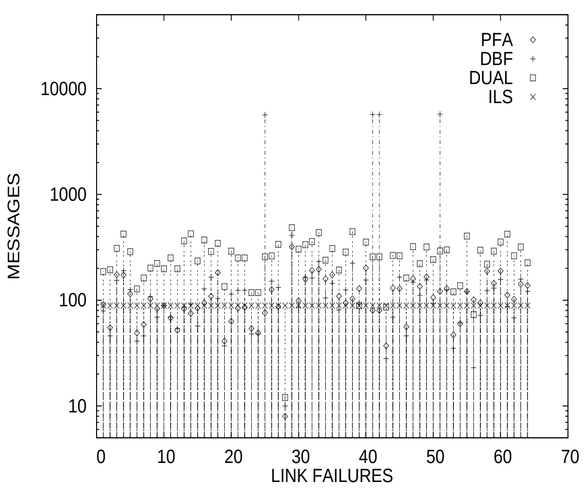
When link (j, k) fails, nodes j and k send update messages to their neighboring nodes as shown in Figure 3(b). In this example, node k is forced to report an infinite distance to j as nodes b and i have reported node k as part of their path to destination j. Node b processes node k's update and selects link (b, j) to destination j. This is because of step(2) of WRP which forces node b to purge any path to node j involving node k. Also, when i gets node k's update message, i updates its distance table entry through neighbor k and checks for the possible paths to destination j through any other neighboring nodes. Thus, a node examines the available paths through its other neighboring nodes and updates the distance and the routing table entries accordingly. This results in the selection of the link (i, j) to the destination j (Figure 3(c)). When node i receives neighbor b's update reporting an infinite distance, node i does not have to update its routing table as it already has correct path information (Figure 3(d)). Similarly, updates sent by node k reporting a distance of 11 to destination j will not affect the path information of nodes i and b. This illustrates how step(2) of WRP helps in the reduction of the formation of temporary loops in the explicit paths.

3 Simulation Results

To gain insight into the average-case performance of WRP in a dynamic environment, we have simulated its operation using an actor-based, discrete-event simulation language called *Drama* [15], together with a network simulation library. Link failures and recoveries are simulated by sending link status message to the nodes at the end points of the appropriate links. Node failures can be treated as all links connecting to that node going down at the same time and the link cost changes can be treated as a link failing and recovering with a new link cost. The connectivity of a mobile node is said to be lost when a node does not hear from a mobile node for a certain period of time. The connectivity with a node will be reestablished when a node hears from a mobile node again. Mobility is modeled as an arbitrary set of failures and recoveries of a mobile node at random points in time. All simulations are done assuming unit propagation time and zero packet processing time at each node. If a mobile node fails when the packets are in transit, the packets are assumed to get dropped.

Our goal is to compare the performance of WRP against the performance of routing protocols based on DBF, DUAL, and ILS. To reduce the complexity of the simulation, we have eliminated those features of the protocols that were common to all; these features concern the reliable transmission of updates over unreliable links, and the identification of neighbors. Accordingly, our simulation assumed that, for any of the protocols simulated, any update message sent over an operational link is received correctly, and that a node always receives enough user messages to know that it continues to have connectivity with a neighbor. According to these assumptions, there is no need to account for acknowledgments, retransmissions of updates, or periodic transmissions of update messages.

However, our intent in running the simulations was to obtain insight on the comparative overhead of different protocols that necessarily require the transmission of acknowledgements to update messages. We approached this problem in the following manner: In a packet radio network, the same update messages sent by a node is received by all its neighbors i.e., each update message is broadcast to a node's neighbors. However, to guarantee the reliable transmission of updates, each neighbor must send an acknowledgement to the sender of the update. Therefore, under the assumption that no errors or collisions occur in the network channel, counting the number of acknowledgements received for a single update broadcast to all neighbors is much the same as counting the number



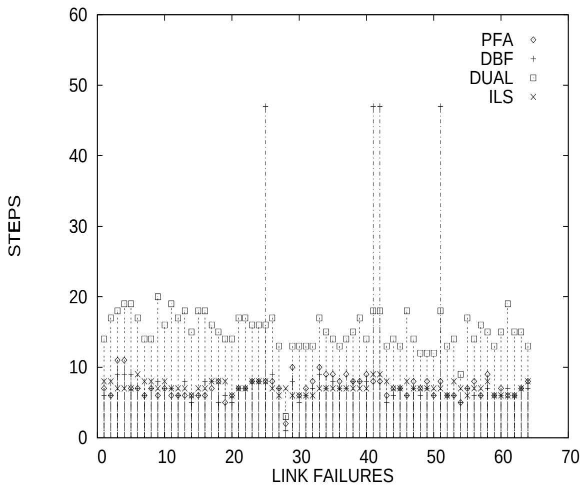
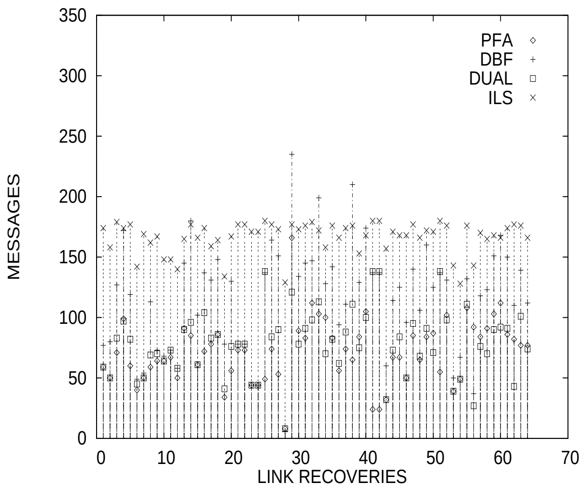


Figure 4: ARPANET Link Failure



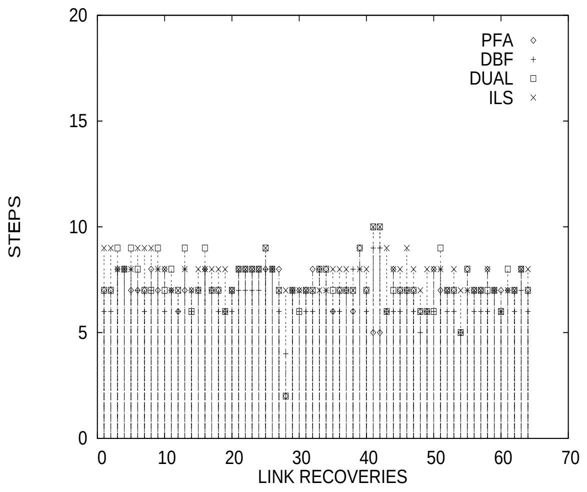


Figure 5: ARPANET Link Recovery

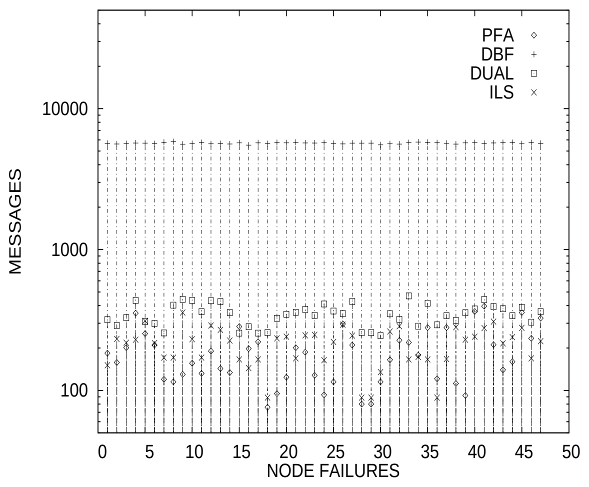
of updates sent by a node to its neighbors on a point-to-point basis and with no acknowledgements—the two counts differ only by one. Accordingly, we simulated the routing protocols' operation in a packet-radio network using the same point-to-point links typical of wireline networks. The message count obtained from the simulation runs is not the exact number of updates and acknowledgements sent by each protocol, but accurately reflects the relative differences among protocols.

The resulting simplified version of WRP we simulated is called "path finding algorithm" (PFA), and is the same basic algorithm first described in [13]. Similarly, ILS, DBF, and DUAL correspond to the ideal case of the best protocols that could be designed based on these algorithms.

To simulate the routing algorithm, a node receives a packet and responds to it by running the routing algorithm, queueing the outgoing packets and processing the updates one at a time in the order in which they arrive. Drama's internals ensure that all the packets at a given time are processed before new updates are generated.

The simulations were run on several network topologies such as *Los-Nettos*, *Nsfnet* and *Arpanet*. We chose these topologies to compare the performance of routing algorithms for well-known cases given that we cannot sample a large enough number of networks to make statistically justifiable statements about how an algorithm scales with network parameters. The los-nettos topology has 11 nodes, a diameter of 4 hops, and each node has at most four neighbors. The Nsfnet topology has 13 nodes, a diameter of 4 hops, and each node has at most 4 neighbors. The ARPANET topology has 57 nodes, a diameter of 8 hops, and each node has a maximum of four neighbors.

For the routing algorithms under consideration, there is only one shortest path between a source and a destination pair and we do not consider null paths from a node to itself. Data are collected for a large number of topology changes to determine statistical distribution. The statistics has been collected after each failure and recovery of a link. To obtain the average figures, we make each link (or node) in the network fail and count the number of steps and messages needed for each algorithm to converge. Then the same link (node) is made to recover and the process is repeated. The average is taken over all failures and recoveries. Again, this message count is not exact, but the relative difference from one



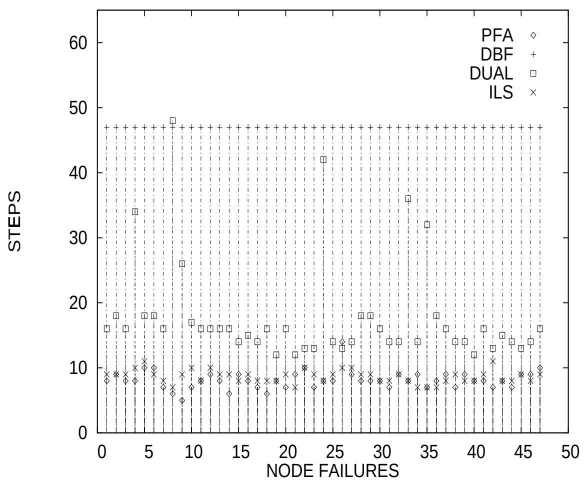
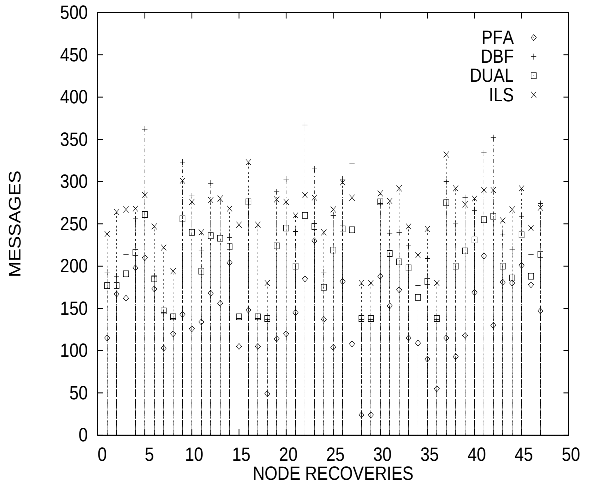


Figure 6: ARPANET Node Failure



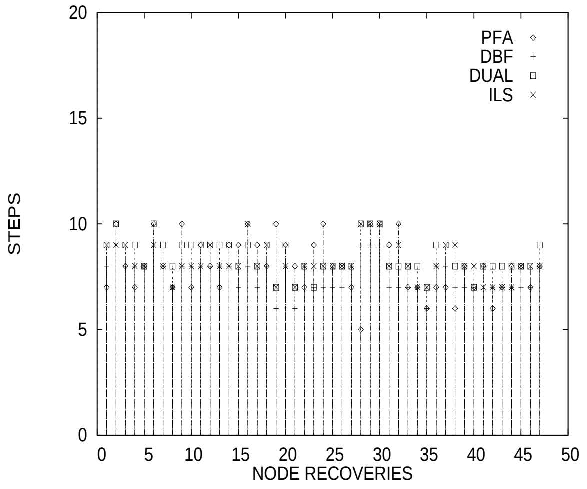


Figure 7: ARPANET Node Recovery

protocol to another is accurate.

3.1 Total Response to a Single Resource Change

The graphs in Figures 4 and 5 depict the number of messages exchanged and the number of steps required before PFA, DBF, DUAL, and ILS converge for every link failing and recovering in the ARPANET topology. We focus more on the results for the ARPANET topology, because of its larger size. Similar graphs for every node failing and recovering are given in Figures 6 and 7 respectively. All topology changes are performed one at a time and the algorithms were allowed to converge after each such change before the next resource change occurs. The ordinates of the graphs represent the identifiers of the links and the nodes while the data points show the number of messages exchanged after each resource change (graphs on the left hand side) and the number of steps needed for convergence (graphs on the right hand side) in each of these figures.

For a single resource failure, PFA outperforms DUAL. This is because, PFA does not use an internodal coordination mechanism that spans several hops to achieve loop freedom. The performance of PFA is comparable to that of ILS after resource failures. The performance of PFA and DUAL is much better than that of ILS after resource recoveries. The counting-to-infinity problem of DBF can be clearly seen in both resource failures and resource recoveries. Given that both resource recoveries and failures will occur in the WRP, PFA offers the best total response to single topology changes, in terms of both update messages and time required to obtain correct routing tables after a topology change.

3.2 Dynamics with Mobile Nodes

We incorporated mobility to the existing fixed network topology by making the links fail and come back up arbitrarily at random points in time. The network is assumed to be fully connected with potential links. At startup, the topology is initialized to some well known topology such as *los-nettos*, *Nsfnet* or *ARPANET*. After initialization, to simulate the movement of a node, a node is assumed to have failed at its previous location and reappear in its new location. Node failure is simulated as all the links associated with that node going down at the same time. The gradual movement of a node from one location to another is simulated by means of link failures and additions. When a link fails, it can be assumed that a node is no longer in the neighborhood of its previous neighbor. The addition of a new link is viewed as a movement of a node wherein, a node reappears in the new neighborhood.

The links are chosen at random from the set of all the existing links in the fully connected network. Selecting any particular link is equally likely. The probability of a link failing or recovering is also equally likely. We also have imposed an additional condition in our simulations that a node at any given time cannot have more than x neighbors. Here, x indicates the degree of the node. This condition is imposed in order to make sure that all the links pertaining to one node alone will not be active. This helps in simulating the mobility more closely.

The average number of messages and the average message length for each of these algorithms are obtained by varying the interarrival time between two events (Figures 8–10). An event can be either a link failure or a link recovery. For the purpose of event generation, we consider a fully connected topology and start off with a given initial topology. Since any random link can fail or recover at any time, our model simulates mobility closely.

The above results indicate that the routing algorithm of WRP outperforms all other algorithms which we have simulated, namely, DBF, DUAL and ILS. We were not able to simulate ILS algorithm for ARPANET topology due to limited resources. The statistics about the average number of messages and the average message length have been collected for all the above mentioned topologies for all the four algorithms by varying the interarrival time between events (failures and recoveries).

In all cases, the average number of messages for DBF and DUAL are more than that of WRP. This is because, DBF suffers from counting-to-infinity problem and DUAL uses an interneighbor coordination mechanism to achieve loop-freedom and this synchronization mechanism spans the entire diameter of the network. ILS sends maximum number of messages since the complete topology information has to be transmitted to each node every time the topology changes.

The average length of each message is the highest in DUAL as compared to all other algorithms. The average message length in case of ILS is almost constant since it always sends the complete topology information. Even though we do not have simulation results for ILS in case of ARPANET topology, we can extrapolate the results from the other two network topologies and can expect similar behavior for ARPANET topology also.

4 Conclusion

A new routing protocol, WRP, for a packet radio network has been presented. This protocol is based on a path-finding algorithm which substantially reduces the number of cases in which routing loops can occur. A mechanism has been proposed for the reliable exchange of update messages. The performance of the routing algorithm in WRP has been compared with that of an ideal topology broadcast algorithm (ILS), DUAL and DBF for highly dynamic environment through simulations. The simulation results show that WRP provides about 50% improvement in the convergence time as compared to DUAL. The results indicate that WRP is a good alternative for routing in packet radio networks.

References

[1] D. Beyer *et. al.*, “Packet Radio Network Research, Development and Application”, *Proc. SHAPE Conference on Packet Radio*, Amsterdam, 1989.

[2] D. Beyer, “Accomplishments of the DARPA SURAN Program”, *Proc. IEEE MILCOM 90*, Monterey, California, Dec. 1990.

[3] D. Bertsekas and R. Gallager, *Data Networks*, Second Ed. Prentice Hall, Inc. 1992.

[4] C. Cheng, R. Reley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, “A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect”, *ACM Computer Communications Review*, 19 (4), 1989, pp.224–236.

[5] Charles E. Perkins and Pravin Bhagwat, “Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers’”, *ACM SIGCOMM*, Vol.24 (No.4), Oct. 1994, pp.234–244.

[6] M. Scott Corson and Anthony Ephremides, “A Distributed Routing Algorithm for Mobile Wireless Networks”, *ACM J. of Wireless Networks*, Jan. 1995, pp. 61–81.

[7] J.J. Garcia-Luna-Aceves, “A Fail-Safe Routing Algorithm for Multihop Packet-Radio Networks”, *IEEE INFOCOM* April, 1986.

[8] J.J. Garcia-Luna-Aceves, “Loop-Free Routing Using Diffusing Computations”, *IEEE/ACM Trans. Networking*, Vol.1, No. 1, Feb. 1993, pp.130–141.

[9] J. Hagouel, “Issues in Routing for Large and Dynamic Networks”, *IBM Research Report*, RC 9942 (No. 44055), April 1983.

[10] P.A. Humblet, “Another Adaptive Shortest-Path Algorithm”, *IEEE Trans. Comm.*, Vol.39, No.6, June 1991, pp.995–1003.

[11] B.M. Leiner, D.L. Nielson and F.A. Tobagi, *Proc. IEEE*, Packet Radio Networks Special Issue, Jan. 1987.

[12] J. Moy, “OSPF Version 2”, *RFC 1583*, March 1994.

[13] Shree Murthy and J.J. Garcia-Luna-Aceves, “A More Efficient Path-Finding Algorithm”, 28th *Asilomar Conference*, Pacific Groove, CA, Nov. 1994.

[14] B. Rajagopalan and M. Faiman, “A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems,” *Journal of Internetworking: Research and Experience*, Vol. 2, No. 1, March 1991, pp. 51-69.

[15] W. T. Zaumen, “Simulations in Drama”, *Network Information System Center, SRI International*, Menlo Park, California, Jan. 1991.

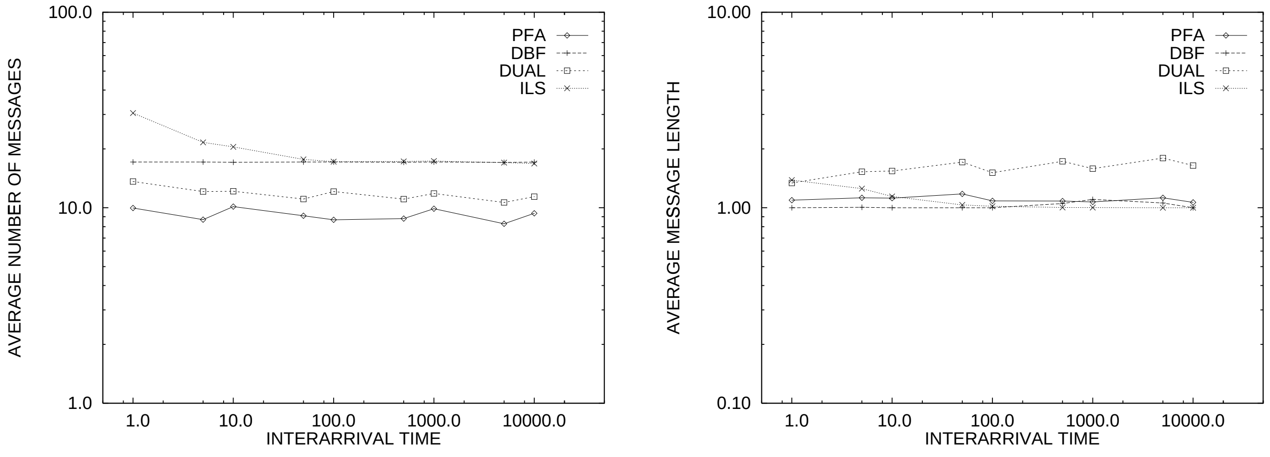


Figure 8: Los-Nettos

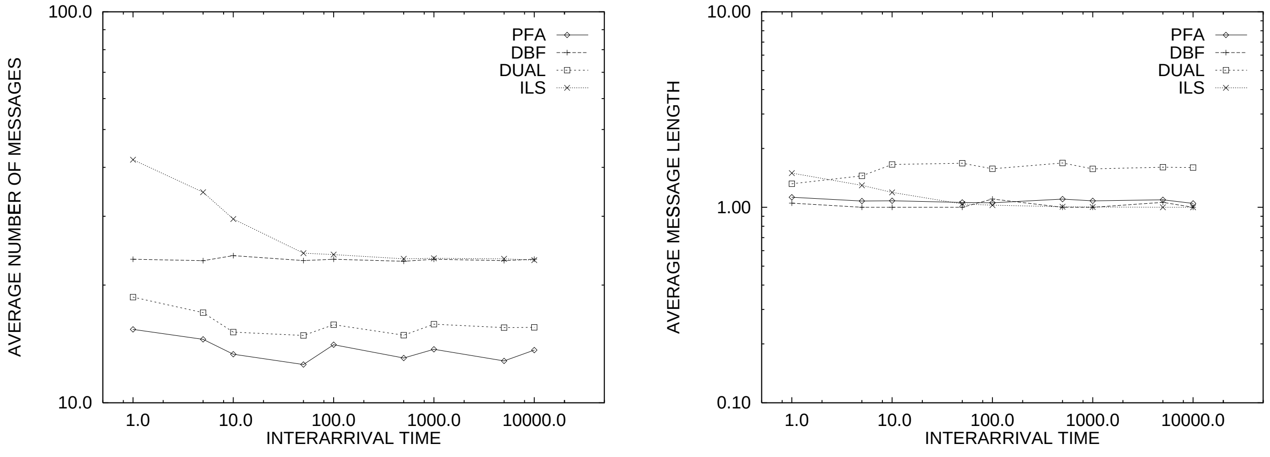


Figure 9: Nsfnet

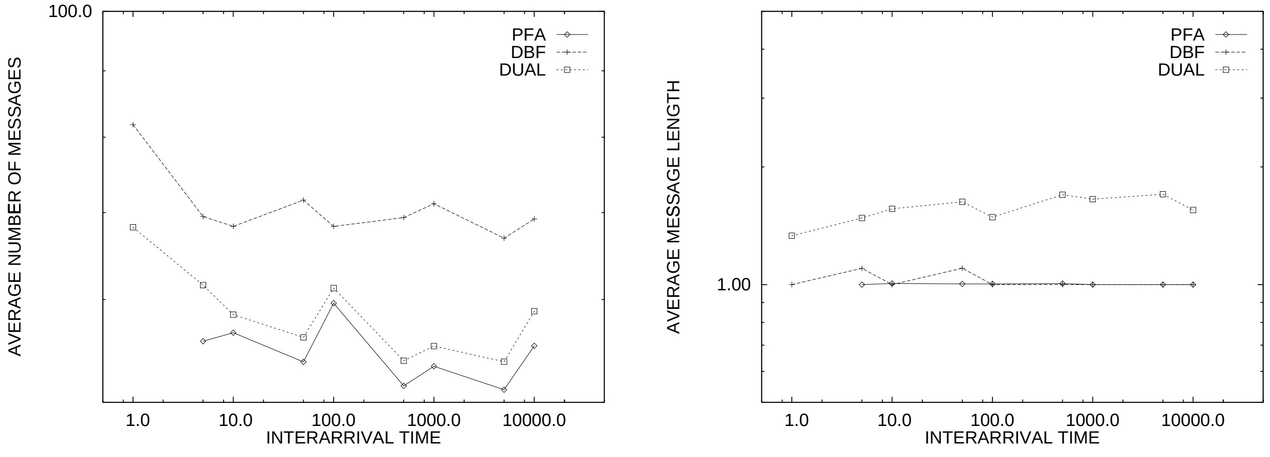


Figure 10: ARPANET